**67. Add Binary**

Given two binary strings a and b, return *their sum as a binary string*.

Input: a = "1010", b = "1011"

Output: "10101"

**Sol:**

**844. Backspace String Compare**

Given two strings s and t, return true *if they are equal when both are typed into empty text editors*. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

**Input:** s = "ab#c", t = "ad#c"

**Output:** true

**Explanation:** Both s and t become "ac".

**Input:** s = "a#c", t = "b"

**Output:** false

**Explanation:** s becomes "c" while t becomes "b".

**Sol:**

**696. Count Binary Substrings**

Given a binary string s, return the number of non-empty substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

**Input:** s = "10101"

**Output:** 4

**Explanation:** There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

**Sol:**

String s = “0011100011”

Here we can divide the string into 3 part ….00111, 111000, 00011

Here in 00111 part we can have 2 binary substring that is min(count of 1, count of 0)

Here in 111000 part we can have 3 binary substring that is min(count of 1, count of 0)

Here in 00011 part we can have 2 binary substring that is min(count of 1, count of 0)

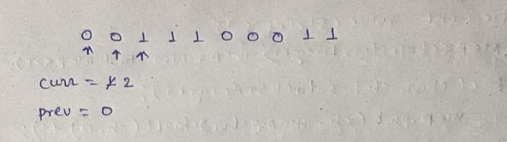
So over all no of binary substring will be 2 + 3 + 2 = 7

**Steps:**

Take current and prev 2 variables.

Current = 1 as it will represent the no of char in the block

Prev = 0



We will increment the current counter if we find the same element as previous one. Here in the case when control is at 0th index current is 1. When control goes to 1st index current will 2. When control goes to index 3 then we will stop as 3rd index element is not same as 2nd index element.

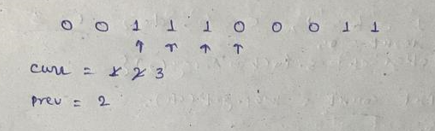
Now we will find out the Min(curr, prev) ….min(2, 0) which will return 0

Add this result value to result. So as f now result will be 0 only.

And put curr = 1

And prev = curr

Then for next block again we need to put the current count to 1 and prev = current

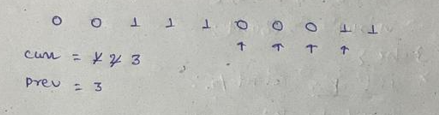


Here 3rd, 4th, 5th element is same so current count will increase from 1 to 2 to 3. When we go to the 5th index then it is not equal to 4th index element. So we stop there.

Now we will find the min(3, 2) which is 2..add this to result so result will be 2 now.

And put current = 1

And prev = current.

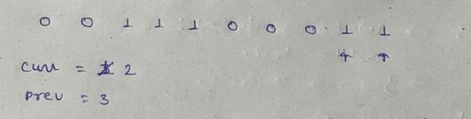


Here 5th, 6th, 7th element is same so current count will increase from 1 to 2 to 3. When we go to the 8th index then it is not same as 7th index element. So we stop there.

Now we will find the min(3, 3) which will return 3…add this to result..so result will be 2 + 3 = 5 now.

And put curr = 1

And prev = curr



Here 8th and 9th element are same so current counter will go from 1 to 2. Once we reach to the end loop will break. So we need to add one min cond outside of the loop.

So now we will find min(2, 3) which will return 2 …. Add this to result..so result will be 5 + 2 = 7

And this is result of the binary substring count.

**1002. Find Common Characters**

Given a string array words, return *an array of all characters that show up in all strings within the*words*(including duplicates)*. You may return the answer in **any order**.

**Input:** words = ["bella","label","roller"]

**Output:** ["e","l","l"]

**Sol:**

int[] common = new int[26]; initialize it with max values

common = [2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647, 2147483647]

Now iterate the string array and count the freq of each char of the word and store it in freq[]

**Now fre count for the bella is**

[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Now perform

common[i] = Math.*min*(common[i], freq[i]);

common becomes

[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

**Now find the freq count for the label**

[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Now perform

common[i] = Math.*min*(common[i], freq[i]);

common becomes

[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

**Now find the freq count for the roller**

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]

Now perform

common[i] = Math.*min*(common[i], freq[i]);

common becomes

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

And this is the common char count for all the three words.

**205. Isomorphic Strings**

Given two strings s and t, *determine if they are isomorphic*.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

**Input:** s = "egg", t = "add"

**Output:** true

**Input:** s = "foo", t = "bar"

**Output:** false

**Sol:**

Map s[i] to t[i] …..Create a map…..iterate the s string one by one. ..if map contains key s[i] then find out value of that key and if value is not equal to t[i] then return false. If map does not contain key but value is present then return false…else add the s[i] as key and t[i] as values in map.

**392. Is Subsequence**

Given two strings s and t, return true if s is a ***subsequence*** of t, or false otherwise.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

**Input:** s = "abc", t = "ahbgdc"

**Output:** true

Sol:

**14. Longest Common Prefix**

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Input:** strs = ["flower","flow","flight"]

**Output:** "fl"

**409. Longest Palindrome**

Given a string s which consists of lowercase or uppercase letters, return *the length of the****longest palindrome*** that can be built with those letters.

Letters are **case sensitive**, for example, "Aa" is not considered a palindrome here.

**Input:** s = "abccccdd"

**Output:** 7

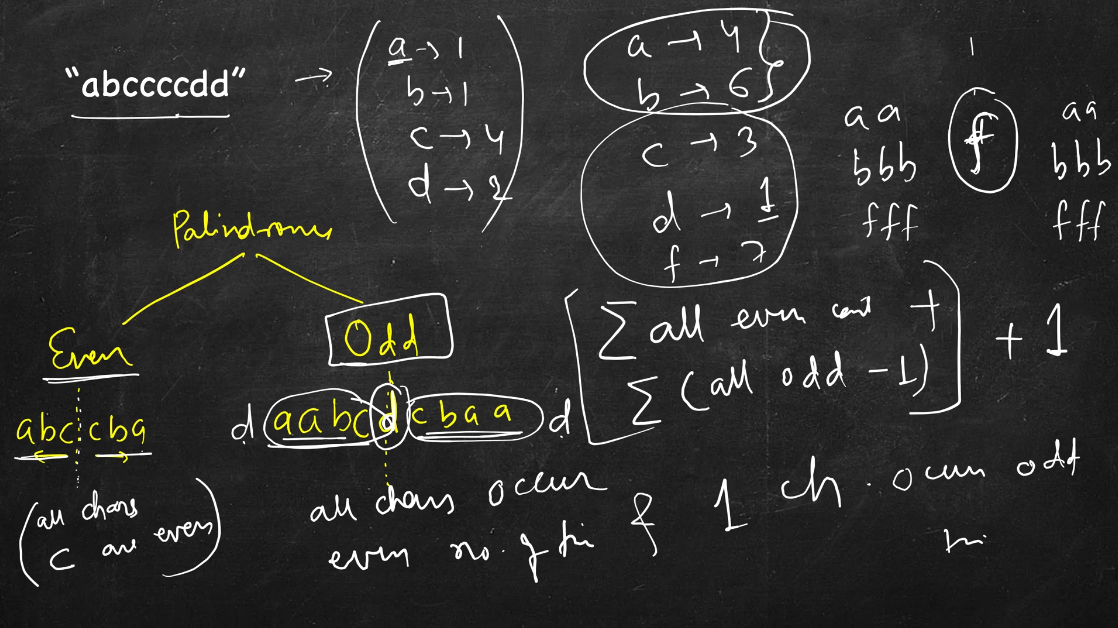
**Explanation:** One longest palindrome that can be built is "dccaccd", whose length is 7.

**Sol:**

Take the map and store the count of each char of the input.

Palindrome can be two type: even length palindrome and odd length palindrome.

If the count of the char is even then we can simply keep same number of char to both side. If it is odd then we will make this char as middle element of the palindrome and reduce its size by 1 so that it will be even length and then we will equally put in both half.



**383. Ransom Note**

Given two strings ransomNote and magazine, return true*if*ransomNote*can be constructed by using the letters from*magazine*and*false*otherwise*.

Each letter in magazine can only be used once in ransomNote.

**Input:** ransomNote = "a", magazine = "b"

**Output:** false

**Input:** ransomNote = "aa", magazine = "ab"

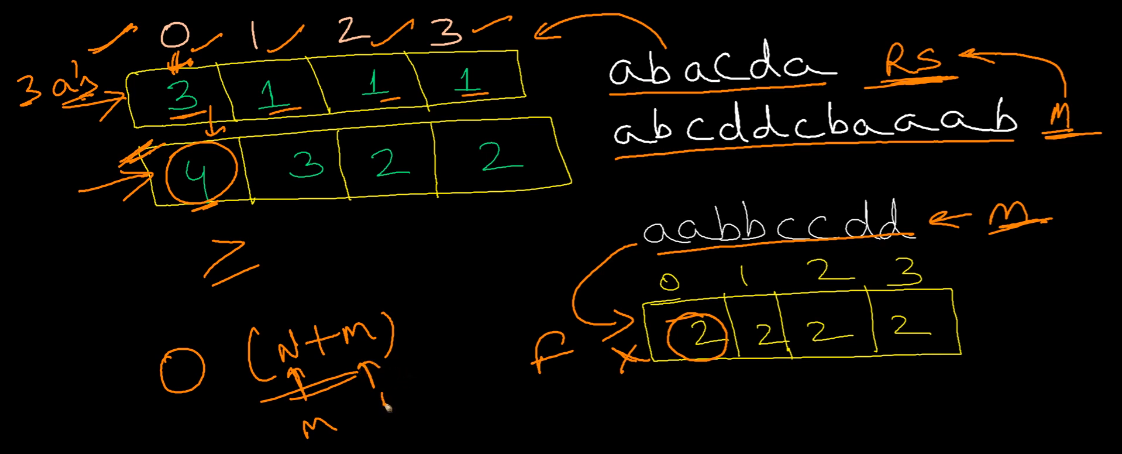
**Output:** false

**Input:** ransomNote = "aa", magazine = "aab"

**Output:** true

**Sol:**

Will take a count of each char of the magazine. While iterating ransom char we will check if char is present in the magazine or not. If it is present, we should go to next char and delete that char so that we will not be able to use that char again.



We need to check one more scenario …if magazine char count is less that ransom note char count then it is not possible to create the ransom note.

**1047. Remove All Adjacent Duplicates In String**

You are given a string s consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

**Input:** s = "abbaca"

**Output:** "ca"

**Input:** s = "azxxzy"

**Output:** "ay"

**Sol:**

1) take one empty stack.

2) iterate the input char one by one and if its top char is equal to current char then we will pop the element from stack.

3) If it is not equal then we will add that char to result char array.

4) iterate the stack and reverse it to get the desired output.

**459. Repeated Substring Pattern**

Given a string s, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

**Input:** s = "abab"

**Output:** true

**Explanation:** It is the substring "ab" twice.

**Sol:**

Here ab is repeated twice. So this is the logic.

1) We will find the string of 1, 2, 3 lengths……and so on…

2) check if the length of the string is divisible by length of substring.

3) If yes we will find out the number how many times.

4) Once we get the repeat count, we will create a string and append the substring that many times.

5) And then we will check if it is equal to original string or not. If it is then we will return true else false.

**917. Reverse Only Letters**

Given a string s, reverse the string according to the following rules:

* All the characters that are not English letters remain in the same position.
* All the English letters (lowercase or uppercase) should be reversed.

Return s*after reversing it*.

**nput:** s = "ab-cd"

**Output:** "dc-ba"

Sol:

**557. Reverse Words in a String III**

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Input:** s = "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Sol:**

Take a pointer i which will point the start index of the word and j will point the last index of word. Iterate the string and when we get the white space chare then we simply reverse the string from start to end index. Once we come out of the loop we will not find any white space so we need to reverse one more time out side loop from start to end index.

**13. Roman to Integer**

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

**Symbol** **Value**

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

**Input:** s = "III"

**Output:** 3

**Explanation:** III = 3.

**796. Rotate String**

Given two strings s and goal, return true *if and only if* s *can become* goal *after some number of****shifts****on* s.

A **shift** on s consists of moving the leftmost character of s to the rightmost position.

* For example, if s = "abcde", then it will be "bcdea" after one shift.
* **Input:** s = "abcde", goal = "cdeab"
* **Output:** true
* **Input:** s = "abcde", goal = "abced"
* **Output:** false

**821. Shortest Distance to a Character**

Given a string s and a character c that occurs in s, return *an array of integers*answer*where*answer.length == s.length*and*answer[i]*is the****distance****from index*i*to the****closest****occurrence of character*c*in*s.

The **distance** between two indices i and j is abs(i - j), where abs is the absolute value function.

**Input:** s = "loveleetcode", c = "e"

**Output:** [3,2,1,0,1,0,0,1,2,2,1,0]

**Explanation:** The character 'e' appears at indices 3, 5, 6, and 11 (0-indexed).

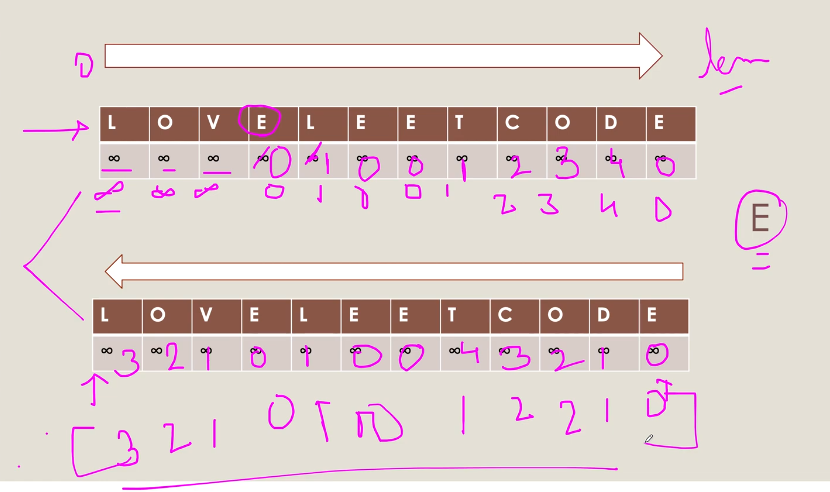
The closest occurrence of 'e' for index 0 is at index 3, so the distance is abs(0 - 3) = 3.

The closest occurrence of 'e' for index 1 is at index 3, so the distance is abs(1 - 3) = 2.

For index 4, there is a tie between the 'e' at index 3 and the 'e' at index 5, but the distance is still the same: abs(4 - 3) == abs(4 - 5) = 1.

The closest occurrence of 'e' for index 8 is at index 6, so the distance is abs(8 - 6) = 2.

**Sol:**



1) Take 2 int array, one for left and one for right and initialize it with int max value.

2) Iterate left to right the given input

a) while iterating if we find the given char then we update the value to 0

b) if char != givenChar then we simply increment the count if it is not int\_max

3) iterate the given array once more, and this time from right to left

a) perform the same operation like in the above steps

4) Minimum of both array for each char will give the result.

**242. Valid Anagram**

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Input:** s = "rat", t = "car"

**Output:** false

**680. Valid Palindrome II**

Given a string s, return true *if the*s*can be palindrome after deleting****at most one****character from it*.

**Input:** s = "aba"

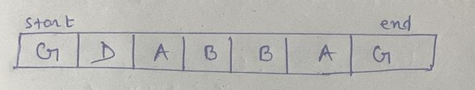
**Output:** true

**Input:** s = "GDABBAG"

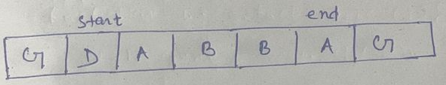
**Output:** true

**Sol:**

Take 2 pointer start and end.



If start and end char are equal then do i++, j- -



If it is not equal then we have 2 conditions

* 1. If we delete D then from start+1 to end string should be palindrome
  2. If we delete A then from start to end-1 string should be palindrome.

If either of one is returning true then we can say that if we delete one char from the string , string will be palindrome.

**290. Word Pattern**

Given a pattern and a string s, find if s follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** word in s.

**Input:** pattern = "abba", s = "dog cat cat dog"

**Output:** true

**Input:** pattern = "abba", s = "dog cat cat fish"

**Output:** false

Sol:

Take a map of char and String. And map patters char to String word. If key is not present in the map we will add into map. If it is present then we will check its value. If its value != current word then return false.